1

A) Briefly disclose problem solving techniques in c programming.

B) Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.

C) Write an algorithm and draw a flowchart that finds the largest of three numbers.

A) Problem Solving Techniques in C Programming

1. **Understand the Problem:** Break down the problem to understand the requirements and constraints. Determine inputs, outputs, and expected behavior.
2. **Plan the Solution:**
   - **Algorithm Design:** Create a step-by-step plan (algorithm) that outlines the process needed to solve the problem.
   - **Pseudocode:** Write the steps in pseudocode to focus on logic without worrying about syntax.
3. **Write the Code:** Convert the algorithm into C code. Pay attention to syntax, data types, and logical structure.
4. **Debugging:** Run the code to test for errors. Use debugging tools or insert print statements to track the program's execution.
5. **Optimization:** Refactor the code to improve efficiency or readability.
6. **Testing:** Test the program with various inputs to ensure it handles all edge cases and behaves as expected.
7. **Documentation:** Comment on the code to explain complex sections and logic. This aids in future maintenance.

B) Algorithm and Flowchart to Calculate the Area of a Rectangle

Algorithm:

1. **Start**
2. **Input** the length and width of the rectangle (L, W).
3. **Calculate** the area using the formula: Area = L × W.
4. **Output** the calculated area.
5. **End**

Flowchart:

- **Start** (Oval)
- **Input Length and Width** (Parallelogram)
- **Calculate Area = L × W** (Rectangle)
- **Output Area** (Parallelogram)
- **End** (Oval)

C) Algorithm and Flowchart to Find the Largest of Three Numbers

Algorithm:

1. **Start**
2. **Input** the three numbers (A, B, C).
3. **If** A > B and A > C, then **Largest = A**.
4. **Else If** B > A and B > C, then **Largest = B**.
5. **Else**, **Largest = C**.
6. **Output** the largest number.
7. **End**

Flowchart:

- **Start** (Oval)
- **Input Three Numbers (A, B, C)** (Parallelogram)
- **Check If A > B and A > C** (Decision Diamond)
    - If **True**, **Largest = A** (Rectangle)
    - If **False**, **Check If B > A and B > C** (Decision Diamond)
        - If **True**, **Largest = B** (Rectangle)
        - If **False**, **Largest = C** (Rectangle)
- **Output Largest** (Parallelogram)
- **End** (Oval)

2

A) Write a program in c including all function of array: strcpy, strcat, strlen, strcmp.

B) With help of example elaborate the purpose of the printf(), scanf() function? How is it used within a C program? Compare with the putchar(), getchar()

A) C Program Demonstrating Array Functions: strcpy, strcat, strlen, strcmp

```c
#include <stdio.h>
#include <string.h>

int main() {
    char str1[50], str2[50], str3[50];
    int len, cmp;

    // Using strcpy to copy one string to another
    strcpy(str1, "Hello");
    printf("str1 after strcpy: %s\n", str1);

    // Using strcat to concatenate two strings
    strcpy(str2, "World");
    strcat(str1, " ");
    strcat(str1, str2);
    printf("str1 after strcat: %s\n", str1);

    // Using strlen to find the length of the string
    len = strlen(str1);
    printf("Length of str1: %d\n", len);

    // Using strcmp to compare two strings
    strcpy(str3, "Hello World");
    cmp = strcmp(str1, str3);

    if (cmp == 0) {
        printf("str1 and str3 are equal.\n");
    } else {
        printf("str1 and str3 are not equal.\n");
    }

    return 0;
}
```

Explanation:

1. **strcpy(str1, "Hello");**: Copies the string "Hello" into str1.
2. **strcat(str1, " "); strcat(str1, str2);**: Concatenates a space and then the contents of str2 ("World") to str1.
3. **strlen(str1);**: Calculates the length of the string str1.
4. **strcmp(str1, str3);**: Compares str1 with str3. If they are identical, it returns 0.

3

A) Write a C Program to compute the sum of even numbers and the sum of odd numbers using a function.

B) Write a C program using do while loop to add all the numbers entered by a user until user enters 0.

A) C Program to Compute the Sum of Even and Odd Numbers Using a Function

```c
#include <stdio.h>

// Function to compute the sum of even and odd numbers
void computeSums(int arr[], int size, int *evenSum, int *oddSum) {
    *evenSum = 0;
    *oddSum = 0;
    for(int i = 0; i < size; i++) {
        if(arr[i] % 2 == 0) {
            *evenSum += arr[i];
        } else {
            *oddSum += arr[i];
        }
    }
}

int main() {
    int size, evenSum, oddSum;

    // Input the number of elements
    printf("Enter the number of elements: ");
    scanf("%d", &size);

    int arr[size];

    // Input the elements
    printf("Enter the elements:\n");
    for(int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    // Call function to compute sums
    computeSums(arr, size, &evenSum, &oddSum);

    // Output the results
    printf("Sum of even numbers: %d\n", evenSum);
    printf("Sum of odd numbers: %d\n", oddSum);

    return 0;
}
```

Explanation:

- **computeSums(int arr[], int size, int *evenSum, int *oddSum)**: This function takes an array and its size, then computes the sum of even and odd numbers, storing the results in the variables evenSum and oddSum.
- ***evenSum += arr[i];**: Adds the current even number to evenSum.
- ***oddSum += arr[i];**: Adds the current odd number to oddSum.

B) C Program Using do-while Loop to Add All Numbers Until User Enters 0

```c
#include <stdio.h>

int main() {
    int num, sum = 0;

    // Using do-while loop to add numbers until user enters 0
    do {
        printf("Enter a number (0 to stop): ");
        scanf("%d", &num);
        sum += num;
    } while(num != 0);

    // Output the total sum
    printf("Total sum: %d\n", sum);

    return 0;
}
```

Explanation:

- **do { ... } while(num != 0);**: The loop ensures that the code inside the block runs at least once and continues until the user enters 0.
- **sum += num;**: Adds the entered number to sum.
- **scanf("%d", &num);**: Reads the number entered by the user.

This program keeps asking the user for numbers, adds them up, and stops when the user enters 0, printing the total sum at the end.

4

A) Discuss the two way selection (if, if-else, nested if-else, cascaded if else) in C language with syntax and program.

B) using switch case concept, write a c program that asks a user to type a character from the following R, G, B and displays Red or Green or Blue or Unknown character depending to the entered letter

A) Two-Way Selection in C Language

1. if Statement:

- **Syntax:**

if (condition) {

   // Statements to execute if the condition is true

}

Example Program:

```
#include <stdio.h>

int main() {
    int num = 10;

    if (num > 0) {
        printf("The number is positive.\n");
    }

    return 0;
}
```

Explanation: If the condition num > 0 is true, the program prints "The number is positive."

2. if-else Statement:

- **Syntax:**

if (condition) {

  // Statements to execute if the condition is true

} else {

  // Statements to execute if the condition is false

}

Example Program:

```c
#include <stdio.h>

int main() {
    int num = -5;

    if (num > 0) {
        printf("The number is positive.\n");
    } else {
        printf("The number is not positive.\n");
    }

    return 0;
}
```

- **Explanation**: If the condition num > 0 is false, the program prints "The number is not positive."

3. Nested if-else Statement:

- **Syntax:**

```c
if (condition1) {
    // Statements to execute if condition1 is true
    if (condition2) {
        // Statements to execute if condition2 is true
    } else {
        // Statements to execute if condition2 is false
    }
} else {
    // Statements to execute if condition1 is false
}
```

- **Example Program:**

```c
#include <stdio.h>

int main() {
    int num = 5;

    if (num > 0) {
        if (num % 2 == 0) {
            printf("The number is positive and even.\n");
        } else {
            printf("The number is positive and odd.\n");
        }
    } else {
        printf("The number is not positive.\n");
    }

    return 0;
}
```

- **Explanation**: This program checks if num is positive, then further checks if it is even or odd using nested if-else statements.

4. Cascaded if-else (else if Ladder):

- **Syntax:**

```c
if (condition1) {
    // Statements to execute if condition1 is true
} else if (condition2) {
    // Statements to execute if condition2 is true
} else if (condition3) {
    // Statements to execute if condition3 is true
} else {
    // Statements to execute if all conditions are false
}
```

Example Program:

```c
#include <stdio.h>

int main() {
    int num = 0;

    if (num > 0) {
        printf("The number is positive.\n");
    } else if (num < 0) {
        printf("The number is negative.\n");
    } else {
        printf("The number is zero.\n");
    }

    return 0;
}
```

- **Explanation**: The program uses cascaded if-else statements to check multiple conditions and print the appropriate message.

B) C Program Using switch-case to Display a Color Based on Input

```c
#include <stdio.h>

int main() {
    char ch;

    // Prompt the user to enter a character
    printf("Enter a character (R, G, B): ");
    scanf("%c", &ch);

    // Use switch-case to determine the color
    switch (ch) {
        case 'R':
        case 'r':
            printf("Red\n");
            break;
        case 'G':
        case 'g':
            printf("Green\n");
            break;
        case 'B':
        case 'b':
            printf("Blue\n");
            break;
        default:
            printf("Unknown character\n");
            break;
    }

    return 0;
}
```

Explanation:

- **switch (ch)**: Switches control based on the value of ch.
- **case 'R':**: If the character is 'R', it prints "Red".
- **case 'G':**: If the character is 'G', it prints "Green".
- **case 'B':**: If the character is 'B', it prints "Blue".
- **default:**: If the character doesn't match any case, it prints "Unknown character".

This program handles both uppercase and lowercase inputs (e.g., 'R' or 'r') and responds accordingly.

5

A) Describe the structure of a function with the help of example.

B) Write a program in c using structure to enter five students and four marks of students and calculate the total return the students details

C) Write a Program to find the factorial of a number.

A) Structure of a Function in C

A function in C consists of four main components:

1. **Return Type**: Specifies the type of value that the function returns (e.g., int, float, void).
2. **Function Name**: The identifier used to call the function.
3. **Parameters (Arguments)**: Input values that the function requires (can be zero or more).
4. **Function Body**: The block of code that performs the task and may include a return statement.

Example of a Function Structure:

```c
#include <stdio.h>

// Function to add two numbers
int add(int a, int b) {
    int sum;             // Local variable
    sum = a + b;         // Logic to calculate sum
    return sum;          // Returning the result
}

int main() {
    int result;

    // Function call with arguments 10 and 20
    result = add(10, 20);

    printf("Sum: %d\n", result);

    return 0;
}
```

Explanation:

- **int add(int a, int b):** The function add takes two integer parameters a and b and returns an integer (int).
- **int sum;:** A local variable inside the function to store the result.
- **return sum;:** The return statement sends the result back to the calling function.
- **result = add(10, 20);:** The function add is called, and the result is stored in result.

B) C Program Using Structure to Enter Five Students' Details and Calculate Total Marks

```c
#include <stdio.h>


// Structure to hold student details
struct Student {
    char name[50];
    int marks[4];
    int total;
};

int main() {
    struct Student students[5];  // Array of 5 students

    // Input details for each student
    for (int i = 0; i < 5; i++) {
        printf("Enter name of student %d: ", i + 1);
        scanf("%s", students[i].name);

        students[i].total = 0; // Initialize total to 0

        printf("Enter 4 marks for %s:\n", students[i].name);
        for (int j = 0; j < 4; j++) {
            printf("Mark %d: ", j + 1);
            scanf("%d", &students[i].marks[j]);
            students[i].total += students[i].marks[j];  // Calculate total marks
        }
    }

    // Output student details
    printf("\nStudent Details:\n");
    for (int i = 0; i < 5; i++) {
        printf("Name: %s\n", students[i].name);
        printf("Marks: %d, %d, %d, %d\n", students[i].marks[0],
students[i].marks[1], students[i].marks[2], students[i].marks[3]);
        printf("Total Marks: %d\n", students[i].total);
printf("------------------\n");
}
return 0;
}
```

Explanation:

- **struct Student**: Defines a structure with fields for the student's name, marks (an array of 4 integers), and total marks.
- **students[5]**: An array of 5 Student structures to store details for 5 students.
- **students[i].total += students[i].marks[j];**: Accumulates the total marks for each student.

C) C Program to Find the Factorial of a Number

```c
#include <stdio.h>

// Function to calculate factorial
int factorial(int n) {
    if (n == 0) {
        return 1;  // Base case: 0! = 1
    } else {
        return n * factorial(n - 1);  // Recursive case: n! = n * (n-1)!
    }
}

int main() {
    int num;
    // Input from the user
    printf("Enter a number: ");
    scanf("%d", &num);

    // Calculate factorial and print the result
    printf("Factorial of %d is %d\n", num, factorial(num));

    return 0;
}
```

Explanation:

- **int factorial(int n)**: A recursive function that calculates the factorial of n.
- **if (n == 0)**: The base case for recursion. The factorial of 0 is defined as 1.
- **return n * factorial(n - 1);**: The recursive call that multiplies n by the factorial of n-1.

This program calculates the factorial of a number entered by the user and prints the result.

6

a) Discuss an array? Describe the declaration and initialization of one and two dimensional arrays with example.

b) Write a C program using array that insert the new value in existing numbers.

A) Discuss an Array and Describe the Declaration and Initialization of One and Two Dimensional Arrays

Array: An array is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow you to store multiple values under a single variable name, making it easier to manage and process large amounts of data.

Declaration and Initialization of One-Dimensional Array:

- **Syntax:**

data_type array_name[size];

Example:

```c
int numbers[5];  // Declaration of an array of 5 integers
numbers[0] = 10; // Initialization
numbers[1] = 20;
```

```
numbers[2] = 30;
numbers[3] = 40;
numbers[4] = 50;
```

Alternatively:

int numbers[5] = {10, 20, 30, 40, 50}; // Declaration and initialization

Declaration and Initialization of Two-Dimensional Array:

- **Syntax:**

data_type array_name[rows][columns];

Example:

```
int matrix[2][3];  // Declaration of a 2x3 matrix
matrix[0][0] = 1;  // Initialization
matrix[0][1] = 2;
matrix[0][2] = 3;
matrix[1][0] = 4;
matrix[1][1] = 5;
matrix[1][2] = 6;
```

Alternatively:

int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}}; // Declaration and initialization

B) C Program to Insert a New Value in an Existing Array

```
#include <stdio.h>

int main() {
    int arr[100], n, i, pos, new_value;

    // Input size of the array
    printf("Enter number of elements: ");
    scanf("%d", &n);

    // Input elements of the array
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Input position and new value to be inserted
    printf("Enter the position where you want to insert the new value: ");
    scanf("%d", &pos);
    printf("Enter the new value: ");
    scanf("%d", &new_value);

    // Shifting elements to the right
    for (i = n; i >= pos; i--) {
        arr[i] = arr[i - 1];
    }

    // Insert the new value
    arr[pos - 1] = new_value;
    n++;  // Increment array size

    // Print the updated array
    printf("Array after insertion:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

7

Given the following algorithm:

Step 1: start

Step 2: read marks or Percentage

Step 3: if marks >= 80 then grade = A , go to step 7

Step 4: if marks >= 60 and marks < = 80 then grade then Grade =B go to step 7

Step 5: if marks >= 40 and marks <=60 then grade = C go to step 7

Step 6: display failed

Step 7: display grade.

Step 8: stop.

a) Draw a flow chart of that algorithm.

b) Write a program that executes that algorithm.

A) Flowchart for the Given Algorithm

Here's how the flowchart would look:

1. **Start**
2. **Read Marks/Percentage**
3. **Is Marks >= 80?**
   - Yes → **Grade = A** → **Go to Step 7**
   - No → **Next Condition**
4. **Is Marks >= 60 and <= 80?**
   - Yes → **Grade = B** → **Go to Step 7**
   - No → **Next Condition**
5. **Is Marks >= 40 and <= 60?**
   - Yes → **Grade = C** → **Go to Step 7**
   - No → **Display "Failed"**
6. **Display Grade**
7. **Stop**

B) C Program that Executes the Algorithm

```c
#include <stdio.h>

int main() {
    int marks;
    char grade;

    // Step 2: Read marks
    printf("Enter marks: ");
    scanf("%d", &marks);

    // Step 3 to Step 5: Determine grade based on marks
    if (marks >= 80) {
        grade = 'A';
    } else if (marks >= 60 && marks < 80) {
        grade = 'B';
    } else if (marks >= 40 && marks < 60) {
        grade = 'C';
    } else {
        printf("Failed\n");
        return 0;
    }

    // Step 7: Display grade
    printf("Grade: %c\n", grade);

    // Step 8: Stop
    return 0;
}
```

8

Using structure write a c program to store 5 employee's information (name, age, city, sex and employee id).

A) Briefly discuss a function? Describe the structure of a function.

B) Discuss the difference between function declaration and function definition?

C) Write c program to find the area of a circle using function.

D)Write a Program to find the factorial of a number using function.

C Program Using Structure to Store 5 Employees' Information

```
#include <stdio.h>

// Define a structure to hold employee information
struct Employee {
    char name[50];
    int age;
    char city[50];
    char sex;
    int employee_id;
};

int main() {
    struct Employee employees[5];  // Array of 5 employees

    // Input details for each employee
    for (int i = 0; i < 5; i++) {
        printf("Enter details for employee %d:\n", i + 1);
```

```c
        printf("Name: ");
        scanf("%s", employees[i].name);
        printf("Age: ");
        scanf("%d", &employees[i].age);
        printf("City: ");
        scanf("%s", employees[i].city);
        printf("Sex (M/F): ");
        scanf(" %c", &employees[i].sex);
        printf("Employee ID: ");
        scanf("%d", &employees[i].employee_id);
        printf("----------------------\n");
    }

    // Display details of all employees
    printf("\nEmployee Details:\n");
    for (int i = 0; i < 5; i++) {
        printf("Employee %d:\n", i + 1);
        printf("Name: %s\n", employees[i].name);
        printf("Age: %d\n", employees[i].age);
        printf("City: %s\n", employees[i].city);
        printf("Sex: %c\n", employees[i].sex);
        printf("Employee ID: %d\n", employees[i].employee_id);
        printf("----------------------\n");
    }

    return 0;
}
```

8A) Discuss a Function and Describe the Structure of a Function

A **function** in C is a self-contained block of code that performs a specific task. Functions help in breaking down a large program into smaller, manageable, and reusable components.

Structure of a Function:

1. **Return Type**: Specifies the type of value the function returns (e.g., int, float, void).
2. **Function Name**: The identifier used to call the function.
3. **Parameters**: Input values the function requires, enclosed in parentheses.
4. **Function Body**: The code block that defines what the function does.

Example:

```c
#include <stdio.h>

// Function to add two numbers
int add(int a, int b) {
    return a + b;
}

int main() {
    int sum = add(5, 10);
    printf("Sum: %d\n", sum);
    return 0;
}
```

B) Difference Between Function Declaration and Function Definition

1. **Function Declaration**:
   - A function declaration (or prototype) tells the compiler about a function's name, return type, and parameters.
   - It doesn't contain the function's body.
   - **Example**: int add(int, int);
2. **Function Definition**:
   - A function definition includes the actual body of the function where the logic is implemented.
   - It also specifies the return type, function name, and parameters.
   - **Example**: int add(int a, int b) { return a + b; }

C) C Program to Find the Area of a Circle Using a Function

```c
#include <stdio.h>
#define PI 3.14159

// Function to calculate the area of a circle
float calculateArea(float radius) {
    return PI * radius * radius;
}

int main() {
    float radius;

    // Input radius from the user
    printf("Enter the radius of the circle: ");
    scanf("%f", &radius);

    // Calculate and display the area
    printf("Area of the circle: %.2f\n", calculateArea(radius));

    return 0;
}
```

D) Program to Find the Factorial of a Number Using Function

```
#include <stdio.h>

// Function to calculate factorial
int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

int main() {
    int num;

    // Input from the user
    printf("Enter a number: ");
    scanf("%d", &num);

    // Calculate and display factorial
    printf("Factorial of %d is %d\n", num, factorial(num));

    return 0;
}
```

This program defines a recursive function `factorial` that calculates the factorial of a given number and returns the result.

9

Given the following program”

#include<stdio.h>

#include<stdlib.h>

```
void displayMessage(int n);
int main()
{
int a=3;
printf("Enter a value:");
scanf("%d",a);
displayMessage(a);
printf("The function has been called %d times\n");
system("pause");
return 0;
}
void displayMessage(int n)
{
int i;
i=1;
do{
printf("C program\n");
i++;
}while(i<=n);
}
```

The provided program has a few issues that need correction to work as intended. Below, I'll discuss the problems and provide a corrected version of the program.

Problems in the Original Program:

1. **scanf("%d",a);**:
    - The scanf function requires the address of the variable where the input will be stored. The correct way to pass the address of a is to use &a.

2. **printf("The function has been called %d times\n");:**
    - The program attempts to print the number of times the function displayMessage is called, but it doesn't pass any variable to the printf function to do so.
3. **No argument passed to printf("The function has been called %d times\n");:**
    - The code attempts to print how many times the function displayMessage was called, but no corresponding variable or argument is passed to printf.

Corrected Program:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to display a message n times
void displayMessage(int n);

int main() {
    int a;

    printf("Enter a value: ");
    scanf("%d", &a);  // Corrected to use &a to store the input in variable a

    displayMessage(a);

    printf("The function has been called %d times\n", a);  // Corrected to pass
'a' as an argument to printf

    system("pause");
    return 0;
}

void displayMessage(int n) {
    int i = 1;

    do {
        printf("C program\n");
        i++;
    } while (i <= n);
}
```

Explanation of the Corrected Program:

- **scanf("%d", &a);:**

    - This line correctly takes user input and stores it in the variable a.
- **displayMessage(a);**:
    - This function call now works as expected, displaying the message "C program" a times.
- **printf("The function has been called %d times\n", a);**:
    - This line now correctly prints how many times the message has been displayed.

Output Example:

If the user enters the value 3, the output will be:

```
Enter a value: 3
C program
C program
C program
The function has been called 3 times
Press any key to continue . . .
```

The program correctly asks for a value, then prints "C program" the specified number of times, and finally informs the user how many times the function was called.

10

a) Find and correct the errors in the above program.

b) Explain when you will need to use printf () and scanf() in C language?

c) Write the output when a user enters a=2.

d) If a user types 0 in the black screen, what do you think should be the result?

A) Find and Correct the Errors in the Above Program

Here is the corrected version of the program:

```
#include <stdio.h>
#include <stdlib.h>
```

```c
// Function to display a message n times
void displayMessage(int n);

int main() {
    int a;

    printf("Enter a value: ");
    scanf("%d", &a);  // Corrected: Added '&' to store input in 'a'

    displayMessage(a);

    printf("The function has been called %d times\n", a);  // Corrected: Passing
'a' as an argument

    system("pause");
    return 0;
}

void displayMessage(int n) {
    int i = 1;

    do {
        printf("C program\n");
        i++;
    } while (i <= n);
}
```

B) Explain When You Will Need to Use printf() and scanf() in C Language

- **printf():**
  - Used to output (print) information to the console or standard output.
  - It can display strings, numbers, and formatted data.
  - **Example:**

```c
printf("Hello, World!\n");
printf("The value is: %d\n", a);
```

scanf():

- Used to read input from the user or standard input.
- It stores the input in specified variables.
- **Example**

```c
int a;
scanf("%d", &a);  // Reads an integer and stores it in variable 'a'
```

- **When to Use**:
    - Use printf() when you want to display information to the user.
    - Use scanf() when you want to take input from the user.

C) Write the Output When a User Enters a = 2

When the user enters 2, the program will display the following output:

```
Enter a value: 2
C program
C program
The function has been called 2 times
Press any key to continue . . .
```

- **Explanation**:
    - The program reads the value 2 and stores it in a.
    - The displayMessage function is called with a = 2.
    - The function prints "C program" two times.
    - The program then prints that the function has been called two times.

D) If a User Types 0 in the Black Screen, What Do You Think Should Be the Result?

If the user enters 0, the program will display the following output:

Write a c program which prompts the user to enter two positive numbers, and then the program asks the user to press 1 to continue or press any number to exit. If you press one the program asks you to enter operator (+,-,*,%,/) of your choice, then the program performs the operation according to the operator you have chosen and display the result.

Here's a C program that fulfills the requirements:

```c
#include <stdio.h>

int main() {
    int num1, num2;
    char operator;
    int choice;
    int result;

    do {
        // Prompt the user to enter two positive numbers
        printf("Enter the first positive number: ");
        scanf("%d", &num1);
        printf("Enter the second positive number: ");
        scanf("%d", &num2);

        // Ensure the numbers are positive
        if (num1 <= 0 || num2 <= 0) {
            printf("Both numbers must be positive. Please try again.\n");
            continue;
        }

        // Prompt the user to press 1 to continue or any other number to exit
        printf("Press 1 to continue or press any other number to exit: ");
        scanf("%d", &choice);

        if (choice != 1) {
            printf("Exiting the program.\n");
            break;
        }

        // Ask the user to enter an operator
        printf("Enter an operator (+, -, *, /, %%): ");
        scanf(" %c", &operator);

        // Perform the operation based on the chosen operator
        switch (operator) {
            case '+':
                result = num1 + num2;
                printf("Result: %d + %d = %d\n", num1, num2, result);
                break;
            case '-':
                result = num1 - num2;
                printf("Result: %d - %d = %d\n", num1, num2, result);
                break;
            case '*':
                result = num1 * num2;
                printf("Result: %d * %d = %d\n", num1, num2, result);
                break;
            case '/':
                if (num2 != 0) {
                    result = num1 / num2;
```

```c
                printf("Result: %d / %d = %d\n", num1, num2, result);
            } else {
                printf("Division by zero is not allowed.\n");
            }
            break;
        case '%':
            if (num2 != 0) {
                result = num1 % num2;
                printf("Result: %d %% %d = %d\n", num1, num2, result);
            } else {
                printf("Division by zero is not allowed.\n");
            }
            break;
        default:
            printf("Invalid operator. Please try again.\n");
            break;
        }

    } while (choice == 1);

    return 0;
}
```

Explanation:

1. **User Input**:
   - The program starts by asking the user to enter two positive numbers.
2. **Validation**:
   - If the numbers entered are not positive, the program will ask the user to enter the numbers again.
3. **Choice**:
   - The user is asked to press 1 to continue or any other number to exit. If 1 is pressed, the program proceeds; otherwise, it exits.
4. **Operator Selection**:
   - The user is prompted to enter an operator (+, -, *, /, %).
5. **Operation Execution**:
   - The program performs the selected operation based on the operator provided and displays the result.

6. **Loop**:
   - The program loops back to the beginning if the user chooses to continue by pressing 1.

Example Output:

```
Enter the first positive number: 10
Enter the second positive number: 5
Press 1 to continue or press any other number to exit: 1
Enter an operator (+, -, *, /, %): +
Result: 10 + 5 = 15
Press 1 to continue or press any other number to exit: 2
Exiting the program.
```

12

A. Write the following pointer declarations.

a.i. p, a pointer to an integer.

a.ii. char p, a pointer to a character.

a.iii. fp, a pointer to a float.

a.iv. How is a pointer variable different from an ordinary variable?

B. Write a program to add two integers using pointers

12

A. Pointer Declarations

a.i. **Pointer to an integer**:

```
int *p;
```

a.ii. **Pointer to a character**:

```
char *p;
```

a.iii. **Pointer to a float**:

float *fp;

a.iv. **Difference between a Pointer Variable and an Ordinary Variable**:

- **Ordinary Variable**:
  - Stores a specific data value (e.g., an integer, float, character).
  - Example: int a = 10; stores the value 10 in the variable a.
- **Pointer Variable**:
  - Stores the memory address of another variable.
  - It "points" to the location in memory where the actual data is stored.
  - Example: int *p; declares a pointer p that can store the address of an integer variable.

B. Program to Add Two Integers Using Pointers

```c
#include <stdio.h>

int main() {
    int num1, num2, sum;
    int *ptr1, *ptr2;

    // Input two integers
    printf("Enter the first integer: ");
    scanf("%d", &num1);
    printf("Enter the second integer: ");
    scanf("%d", &num2);

    // Assign the addresses of num1 and num2 to ptr1 and ptr2
    ptr1 = &num1;
    ptr2 = &num2;

    // Calculate the sum using pointers
    sum = *ptr1 + *ptr2;

    // Output the result
    printf("Sum of %d and %d is: %d\n", *ptr1, *ptr2, sum);

    return 0;
}
```

Explanation:

1. **Pointer Declaration**:
   - int *ptr1, *ptr2; declares two pointers ptr1 and ptr2 that can point to integers.
2. **Assigning Addresses**:
   - ptr1 = &num1; and ptr2 = &num2; assign the memory addresses of num1 and num2 to the pointers ptr1 and ptr2.
3. **Dereferencing Pointers**:
   - sum = *ptr1 + *ptr2; adds the values pointed to by ptr1 and ptr2.
4. **Output**:
   - The program prints the sum of the two integers using pointers.

Example Output:

```
Enter the first integer: 10
Enter the second integer: 20
Sum of 10 and 20 is: 30
```